تاریخ دریافت مقاله: ۹۶/۰۹/۰۵
تاریخ پذیرش مقاله: ۹۷/۰۴/۰۵

افزایش شتاب الگوریتم ذرات با استفاده از چارچوب کودا

سمیه طاهریان دهکردی*
دانشجوی دکتری گروه مهندسی کامپیوتر، دانشگاه آزاد اسلامی واحد کرمان، کرمان، ایران.
پست الکترونیکی: s.taherian2000@gmail.com

عمید خطیبی بردسیری
استادیار گروه مهندسی کامپیوتر، دانشگاه آزاد اسلامی واحد کرمان، ،کرمان، ایران.
پست الکترونیکی: khatibi_amid@yahoo.com

چکیده

الگوریتم بهینه‌سازی ذرات از جمله روش‌های فراابتکاری است که به‌صورت گسترده در حل مسائل بهینه‌سازی به‌کار گرفته می‌شود. الگوریتم بهینه‌سازی ذرات از رفتار پرندگان در پرواز الگوبرداری شده است و به‌عنوان یک روش فراابتکاری با همگرایی و دقت مناسب شناخته شده است. یکی از چالش‌های مهم الگوریتم‌های فراابتکاری از جمله الگوریتم بهینه‌سازی ذرات زمان اجرای آن در حل مسائل بهینه‌سازی پیچیده است و در این مسائل به علت تعداد ابعاد مسئله و پیچیدگی تابع هدف زمان یافتن جواب بهینه توسط روش‌های فراابتکاری و از جمله الگوریتم ذرات قابل توجه است. این مشکل زمانی بیشتر می‌شود که برای توابع هدف پیچیده نیاز است اندازه جمعیت به قدر کافی بزرگ در نظر گرفته شود تا فضای مسئله پیمایش مناسب شده و نقاط بهینه محاسبه شوند و این مستلزم محاسبات بیشتر بوده و زمان اجرای الگوریتم را افزایش می‌دهد.

ماهیت الگوریتم‌هایی نظیر بهینه‌سازی ذرات موازی است زیرا اعضای جمعیت تا حدود زیادی به‌صورت مستقل فضای جستجوی مسئله را مورد پیمایش قرار می‌دهند.

در این پژوهش یک روش موازی‌سازی جدید برای تسریع محاسبات الگوریتم بهینه‌سازی ذرات ارایه شده است که از ریزهسته‌های پردازنده گرافیکی و در چارچوب کودا استفاده می‌نمایند. روش پیشنهادی یک اشاره‌گر یا آرایه از جمعیت اولیه را از حافظه اصلی به حافظه سراسری منتقل نموده تا زمان انتقال کمینه شود و از طرفی از یک هسته (هسته) برای به روزرسانی سرعت و موقعیت ذرات استفاده می‌نمایند تا سوئیچ‌های بین ریسه[1]ها به حداقل ممکن کاهش و کارایی الگوریتم موازی افزایش یابد. نتایج ارزیابی ما بر روی مجموعه‌ای از توابع ارزیابی نشان می‌دهد حداکثر شتاب به‌دست آمده در روش پیشنهادی و با پردازنده گرافیکی Geforce 710M در حدود ۲۶ برابر پردازنده اصلی است.

واژه‌های کلیدی: موازی‌سازی، واحد پردازش گرافیکی، الگوریتم‌های تکاملی، الگوریتم بهینه‌سازی ذرات.

۱- مقدمه

بهینه‌سازی به معنای دستیابی به بهترین نتیجه تحت شرایط معین است. از نظر ریاضی، یک مسئله بهینه‌سازی،

* نویسندهٔ مسئول

1- thread

شامل یک تابع شایستگی است که مسئله را تحت مجموعه‌ای از محدودیت ها که نشـــان دهنده فضای حل آن هســـتند، توصیف می‌کند. الگوریتم‌هـــای تکاملی[2] یکی از روش‌های موثر و کارآمد برای حل مســـائل بهینه‌ســـازی[3] محسوب می‌شوند. روش کار الگوریتم‌های تکاملی بر اساس اصول تکامل[4] و انتخاب طبیعی[5] مدل‌ســـازی و تعریف شده است. در این الگوریتم‌ها از رفتارهای بین موجودات و فعالیت‌های زیستی و مرتبط با تکامل جهت خلق الگوریتم‌های محاسباتی که قادر است بهینه مسائل را محاسبه نماید، استفاده شده اســـت. تاکنون تعداد زیادی از الگوریتم‌های فراابتکاری و تکاملی برای حل مســـائل بهینه‌سازی ارایه شده است و به دلیل اســـتفاده از روش‌های مبتنی بر تکامل و عدم نیاز به محاســـبه گرادیان به شکل وسیع در حل این مسائل به‌کار گرفته می‌شوند. الگوریتم‌های هوش دسته جمعی یک نمونه از الگوریتم‌های فراابتکاری در نظر گرفته می‌شوند و مزیت این روش‌ها در این است که فضای جستجوی مسئله برای یافتن جواب بهینه به صورت هوشمندانه‌ای مورد جستجو قرار گرفته می‌شـــود. در الگوریتم‌های هوش دسته جمعی اعضای جمعیت از اطلاعات ســـابق خود در فضای مسئله و اطلاعات افراد بهینه جمعیت برای یافتن مسیرهای منتهی بـــه نقاط بهینه اســـتفاده می‌نمایند. الگوریتم بهینه‌ســـازی دســـته جمعی ذرات[6] یک نمونه بارز از روش‌های دســـته جمعی برای حل مسائل بهینه‌سازی است. با وجود کاربرد الگوریتم بهینه‌ســـازی دســـته جمعی ذرات در حل مسائل بهینه‌ســـازی این الگوریتم در مواجهه با فضای پیچیده و اندازه جمعیت بزرگ که برای یافتن جواب بهینه این فضاها الزامی است با کندی اجرا می‌شود.

یکـــی از روش‌هـــای افزایـــش شـــتاب الگوریتم‌هـــای فراابتکاری نظیـــر ذرات اجرای آن‌ها در پردازنده گرافیکی اســـت تا به صورت موازی اجرا شوند.در این پژوهش یک روش جدیـــد مبتنی بـــر معماری پردازنـــده گرافیکی برای شتاب دادن به الگوریتم بهینه‌سازی ذرات ارایه شده است. در روش پیشـــنهادی کل جمعیت ذرات به عنوان یک آرایه و اشـــاره‌گر در حافظه سراســـری پردازنده گرافیکی قرار داده می‌شود و توســـط مجموعه ریسه‌های کودا موقعیت هر ذره که بخشی از آرایه است توسط دستورات الگوریتم ذرات که در قالب هســـته بازنویسی شده‌اند به روزرسانی می‌شـــود تا اعضای جمعیت موازی به سمت جواب بهینه سوق داده شـــوند. در واقع روش پیشنهادی برای هر ذره یک بردار موقعیت و یک بردار ســـرعت در نظر گرفته شده است و برای کاهش سربار زمان انتقال از حافظه پردازنده اصلی بـــه حافظه پردازنده گرافیکی انتقال اعضای جمعیت در یک مرحله انجام شده است. لذا برای این هدف ما در این پژوهش دو بردار سراسری برای موقعیت ذرات و سرعت آن‌هـــا در نظر گرفته و هر ذره در بخشـــی از این بردارها موقعیت‌یابی و مقداردهی اولیه می‌شـــود. ســـپس با انتقال آن‌ها بین دو حافظه از زمان ســـربار ناشی از سوئیچ‌های مکرر کاســـته می‌شود. در واقع تفاوت روش پیشنهادی ما با ســـایر پژوهش‌ها در این است که انتقال اعضای جمعیت در یک مرحله انجام می‌شـــود و از طرفی پژوهش ما فقط شـــامل یک هسته واحد است تا از فراخوانی مکرر هسته‌ها جلوگیری شود و زمان اجرای الگوریتم موازی کاهش یابد. در ســـازوکار پیشنهادی هسته پیشـــنهادی به مولفه‌های بردار موقعیت جمعیت و بردار ســـرعت آن دسترسی دارد و هر ذره را بر روی یک هســـته و با یک ریسهٔ اختصاصی اجرا می‌نماید. هر کدام از ریسه‌ها در روش پیشنهادی یک نسخه از هسته را در حافظه محلی خود نگهداری نموده و آن را بر روی داده‌های خود که در حافظه سراســـری قرار دارند اجرا می‌نمایند.

بـــرای ارزیابـــی الگوریتم پیشـــنهادی از پردازنده‌های گرافیکی شرکت انویدیا و سری جیفورس و مانند بسیاری از پژوهش‌ها در حـــوزه الگوریتم‌های فراابتکاری از توابع ارزیابی برای سنجش سرعت الگوریتم موازی و پیشنهادی استفاده شده است. توابع ارزیابی یا هم‌سنجی[7] را می‌توان به

2- Evolutionary Algorithms
3- Optimization problems
4- Evolution lo
5-Natural selection
6- Swarm Intelligence(SI)
7- Benchmark Function

عنوان تابع هزینه[8] جهت ارزیابی دقت و کارایی الگوریتم‌های فراابتکاری مورد استفاده قرار داد. توابع ارزیابی مجموعه‌ای از توابع ریاضی می‌باشند که هدف نهایی در آن‌ها یافتن کمینه سراسری[9] و عبور از کمینه‌های محلی[10] به‌کار رفته در آن‌ها می‌باشد. توابع ارزیابی یک معیار استاندارد برای سنجش کارایی الگوریتم‌های فراابتکاری محسوب می‌شوند به گونه‌ای که هر الگوریتم فراابتکاری که بتواند با میزان خطای کمتری بهینه‌های سراسری را محاسبه نماید و برای رسیدن به یک آستانه خطا تکرار کمتری نیاز داشته باشد و از طرفی کمتر در بهینه‌های محلی این توابع گرفتار شود الگوریتم دقیق‌تر و هوشمندتری در نظر گرفته می‌شود. توابع ارزیابی یا سنجش دارای روابط ساده یا پیچیده‌ای می‌باشند که معرف سطح دشوار بودن مسئله بهینه‌سازی مورد نظر است. در این توابع هر چقدر تعداد بهینه‌های محلی بیشتر باشد فضای جستجوی مسئله پیچیده‌تر و دشوارتر خواهد بود. توابع ارزیابی را می‌توان در هر فضای چند بعدی در نظر گرفت اما به علت امکان نمایش آن‌ها در فضای دوبعدی و سه‌بعدی در ارزیابی الگوریتم‌ها بیشتر فضای سه‌بعدی مسئله در نظر گرفته می‌شود. از جمله توابع ارزیابی یا همسنجی پر کاربرد می‌توان به توابع ارزیابی Sphere، SumSquare، Rosenbrock، Griewank، Rastrigin و Ackley اشاره نمود. مجموعه داده در الگوریتم‌های فراابتکاری همان اعضای جمعیت اولیه می‌باشند که در مرحله اول آن‌ها را تصادفی ایجاد نموده و با استفاده از تابع ارزیابی مورد ارزیابی قرار داده شده است.

اندازه جمعیت اولیه در میزان زمان اجرای الگوریتم‌های فراابتکاری نقش مهمی دارد و از این رو در این مقاله این داده‌ها به عنوان داده‌های ورودی در نظر گرفته شده‌اند و به نوعی مختصات دکارتی در فضای مسئله می‌باشند.

ادامه مقاله در ۵ بخش سازماندهی می‌شود. در بخش۲ پیشینه تحقیق و موازی سازی الگوریتم ذرات شرح داده می‌شود. در بخش۳ جزئیات الگوریتم پیشنهادی مطرح می‌گردد. عملکرد الگوریتم پیشنهادی و مقایسه آن با چهار تابع همسنجی در بخش٤ ارائه می‌گردد و در نهایت در بخش۵ نتیجه‌گیری انجام می‌شود.

۲- پیش‌زمینه

الگوریتم بهینه‌سازی ذرات، یکی از مهمترین الگوریتم‌های تکاملی مبتنی بر هوش جمعی برای حل مسائل بهینه‌سازی است که در بسیاری از زمینه‌ها مورد استفاده قرار می‌گیرد، اما این الگوریتم با مشکل همگرایی زودرس روبرو است. دلیل اصلی این امر آن است که فرایند بهینه‌سازی در این الگوریتم نیاز به تعداد زیادی از ارزیابی‌های تابع دارد که معمولا به‌صورت موازی و ردیفی اجرا می‌شوند. هدف اصلی این مقاله تحلیل و بررسی پیاده سازی الگوریتم بهینه سازی ذرات بر روی واحد پردازش گرافیکی است، در واقع در این مقاله پیاده‌سازی و بررسی سرعت اجرای الگوریتم ذرات به‌صورت موازی و مبتنی بر چارچوب کودا مورد بررسی و مقایسه قرار گرفته شده است. سپس تلاش شده است شتاب در این روش را تا حدی بهبود بخشیده و روش جدیدی برای افزایش شتاب در الگوریتم ذرات و محیط پردازنده گرافیکی در چارچوب کودا پیشنهاد گردد. نتایج پیاده‌سازی چهار تابع همسنج بر روی پردازنده گرافیکی نشان می‌دهد که کارایی این الگوریتم نسبت به پیاده‌سازی آن به‌صورت ردیفی و با تغییر در اجرای هسته‌ها بسیار افزایش یافته است.

۲-۱ الگوریتم بهینه‌سازی ذرات

الگوریتم بهینه‌سازی دسته جمعی ذرات [۱] یک الگوریتم تکاملی با رویکرد هوش دسته جمعی است که در آن هر یک از اعضای گروه به کمک سایر اعضای گروه می‌تواند به سمت بهینه مسائل حرکت نماید. در این دسته از الگوریتم‌ها یک جزء به تنهایی توانایی اندکی برای حل مسئله دارد اما در تعامل با سایر افراد گروه و به‌ویژه افراد شایسته‌تر که نقش رهبری دسته را بر عهده دارند می‌تواند به نزدیک

8- Cost function
9- Global minimum
10- Local minimum

جواب‌های مسئله همگرا شوند. این الگوریتم براساس رفتارهای اجتماعی و با مطالعه رفتار گروهی پرندگان در یافتن غذا ارایه و مدل‌سازی گردید. در این الگوریتم هر عضو جمعیت دارای دو مولفه سرعت و موقعیت است که توسط بهترین اعضاء و سابقه حرکت عضو فعلی در هر تکرار به روزرسانی می‌شود. در این الگوریتم اعضای جمعیت در قالب ذرات مدل‌سازی می‌شوند و ذرات به کمک اطلاعات قبلی خود و بهترین موقعیت سایر ذرات تلاش می‌کنند به سمت مناطقی که غذای بیشتری دارند حرکت نمایند. با توجه به این‌که هر عضو جمعیت به تنهایی می‌تواند موقعیت خود را به کمک بهترین عضو سراسری و بهترین موقعیت محاسبه نماید لذا می‌توان آن را به‌صورت موازی تسریع داد. یکی از معایب الگوریتم بهینه‌سازی ذرات، نیاز به جمعیتی به اندازه زیاد برای همگرا شدن به جواب‌های دقیق‌تر مسائل بهینه‌سازی است.

در یک مسئله بهینه‌سازی که با یک تابع هدف مدل‌سازی می‌شود در صورتی که پیچیدگی و ابعاد مسئله زیاد شود به همان نسبت نیاز است که تعداد اعضای جمعیت به قدر کافی بزرگ در نظر گرفته شود تا کل فضای جستجوی[11] تابع هدف برای یافتن بهینه مسئله مورد جستجو قرار گیرد. در توابع هدف پیچیده‌تر و چندبعدی این فرآیند نیازمند زمان اجرای بالایی است که در کاربردهای بسیاری نامناسب و نامطلوب است.

یکی از روش‌های افزایش سرعت اجرای الگوریتم‌های تکاملی و از جمله الگوریتم بهینه‌سازی ذرات استفاده از روش‌های موازی‌سازی است و ماهیت موازی الگوریتم ذرات ساختار این الگوریتم را برای موازی‌سازی ایده‌آل نموده است. جهت موازی‌سازی و افزایش سرعت الگوریتم بهینه‌سازی ذرات روش‌های مختلفی وجود دارد که می‌توان به محاسبات تورین[12] و بسته‌های ریسه[13] موازی‌سازی مختص پردازنده اصلی[14] اشاره نمود. هر یک از این روش‌ها معایب خاص خود را دارند. به‌عنوان مثال در فناوری محاسبات تورین هزینه پیاده‌سازی و پیچیدگی بالا باعث می‌شود که این شیوه موازی‌سازی در دسترس عموم نباشد یا در روش موازی‌سازی ریسه‌ای در پردازنده اصلی به علت انتقال متن[15] فرآیندهای مختلف در پردازنده اصلی این روش کارایی بالایی ندارد.

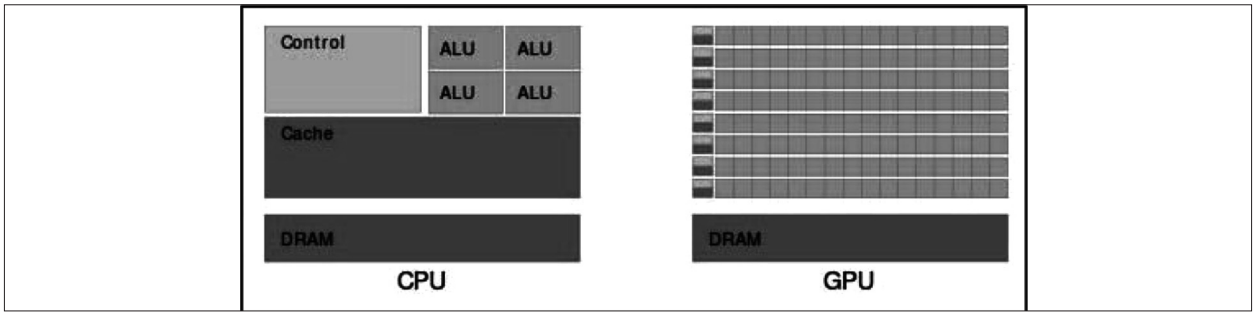
بر خلاف دو روش گفته شده به تازگی شرکت ارایه دهنده پردازنده‌های گرافیکی انویدیا برای موازی‌سازی محاسبات پیچیده در هسته‌های متعدد پردازنده گرافیکی یک چارچوب موازی‌سازی تحت نام کودا[16] ارایه نموده است. چارچوب کودا را می‌توان یک کتابخانه و یک واسط بین برنامه‌نویس و سخت افزار گرافیکی در نظر گرفت که پیچیدگی محاسبات موازی را تا حدود زیادی کاهش می‌دهد و امکانات و دستورات مختلفی را در اختیار برنامه‌نویس قرار می‌دهد تا عملیات موازی‌سازی را انجام دهد[۲]. در واقع دقت و کاهش زمان اجرای الگوریتم‌های تکاملی، هدف اصلی در هنگام حل مسائل بهینه‌سازی است که ارائه راهکار موازی سازی در چارچوب کودا دسترسی به این هدف را امکان‌پذیر کرده است[۳].

۲-۲ موازی سازی در پردازنده‌های گرافیکی

واحدپردازش کارت‌گرافیک[17] را می‌توان پردازنده کارت گرافیک برای انجام عملیات گرافیکی در نظر گرفت. تعداد هسته‌های موجود در پردازنده‌گرافیکی به مراتب بیشتر از پردازنده اصلی است و تعداد هسته‌ها گاهی تا صدها و هزاران عدد می‌رسد.

معماری پردازنده‌گرافیکی به‌صورتی است که بیشتر معماری آن صرف طراحی واحد حساب و منطق[18] آن شده و تمرکز کمتری بر طراحی حافظه اصلی[19] و نهان[20] آن صورت گرفته است چون پردازش‌هایی که در پردازنده گرافیکی انجام می‌شود بیشتر ازنوع پردازش‌های ساده

11- Search space
12- Grid computing
13- Thread
14- Central Processing Unit (CPU)
15- Context switching
16- Compute Unified Device Architecture(CUDA)
17- Graphics Processing Unit (GPU)
18- ALU
19- DRAM
20- Cash



شکل ۱: تفاوت معماری پردازنده اصلی و گرافیکی

ولی با حجم بالا می‌باشند که نیاز است قسمت حساب و منطق گسترش بیشتری نسبت به سایر بخش‌ها داشته باشد[٤و٥].

شکل(۱)، تفاوت سیستم سخت افزاری پردازنده گرافیکی و اصلی را نمایش داده است. همان‌گونه که در این شکل مشاهده می‌کنید تعداد هسته‌های به‌کار رفته در معماری پردازنده‌گرافیکی به مراتب بیشتر از پردازنده اصلی است و در مقابل بخش واحد حساب و منطق پردازنده اصلی نسبت به پردازنده اصلی گسترش بیشتری پیدا کرده است و دلیل آن محاسبات پیچیده‌تر در پردازنده اصلی است[٤]:

از مقایسه معماری پردازنده اصلی و گرافیکی می‌توان به نکات زیر پی برد:

• تعداد هسته‌های پردازنده گرافیکی بسیار بیشتر از پردازنده اصلی است و این نشان می‌دهد که معماری پردازنده گرافیکی و پردازنده اصلی به ترتیب برای محاسبات بر روی داده‌های بزرگ و کوچک مناسب است.

• اندازه هسته‌های پردازنده گرافیکی کوچکتر از پردازنده اصلی است و این نشان دهنده آن است که پردازنده گرافیکی برای محاسبات حجیم اما ساده بهینه‌سازی شده که در پردازنده اصلی عکس این مطلب درست است.

• بخش زیادی از معماری پردازنده اصلی به معماری حافظه نهان اختصاص داده شده است و این در حالی است که در پردازنده گرافیکی حافظه نهان نقش بسیار کلیدی ندارد و علت آن این است که هر هسته به شکل اختصاصی عمل می‌کند و کمتر انتقال متن بین هسته‌های مختلف در پردازنده گرافیکی اتفاق می‌افتد. لذا داده‌ها در دسترس پردازنده می‌باشند اما در پردازنده اصلی به علت تعداد کمک هسته‌ها انتقال متن مرتباً اتفاق می‌افتد و نیاز است داده‌های میانی در حافظه سریع مانند حافظه نهان ذخیره سازی شوند تا در انتقال متن بعدی به داده‌های خود دسترسی سریع داشته باشد.

• بخش کنترل در معماری پردازنده اصلی بزرگتر از پردازنده گرافیکی می‌باشد و دلیل آن پیچیدگی بالای معماری پردازنده اصلی در مقایسه با پردازنده گرافیکی است که نیاز است بخش کنترل آن نیز دقیقاً طراحی شود تا پردازنده اصلی به‌خوبی کنترل و مدیریت شود.

۲-۲-۱ اجرای موازی برنامه‌ها در پردازنده گرافیکی

هر برنامه موازی برای اجرا در پردازنده گرافیکی به دو بخش اصلی ذیل تقسیم می‌شود:

• کد میزبان[۲۱](قابل اجرا در پردازنده اصلی)

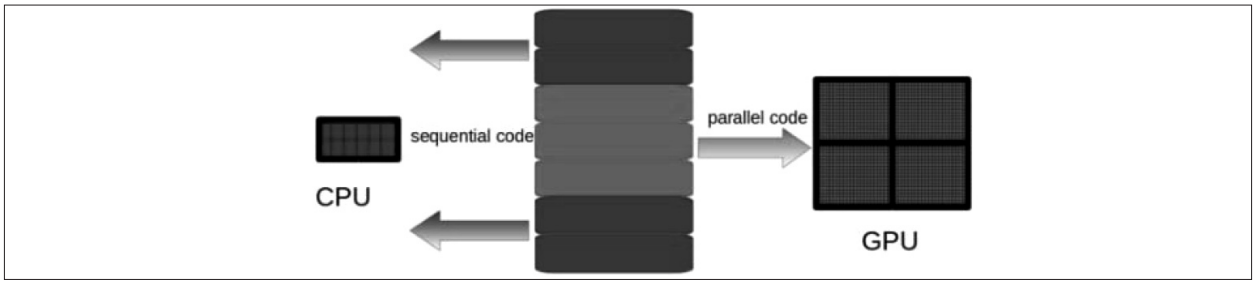
• کد دستگاه[۲۲](قابل اجرا در پردازنده گرافیکی)

کد میزبان در واقع بخش ردیفی و غیرموازی کد برنامه است که فقط می‌تواند در پردازنده اصلی اجرا شود و این در صورتی است که کد دستگاه بخش موازی کد برنامه را تشکیل می‌دهد و فقط قابل اجرا در پردازنده گرافیکی است.

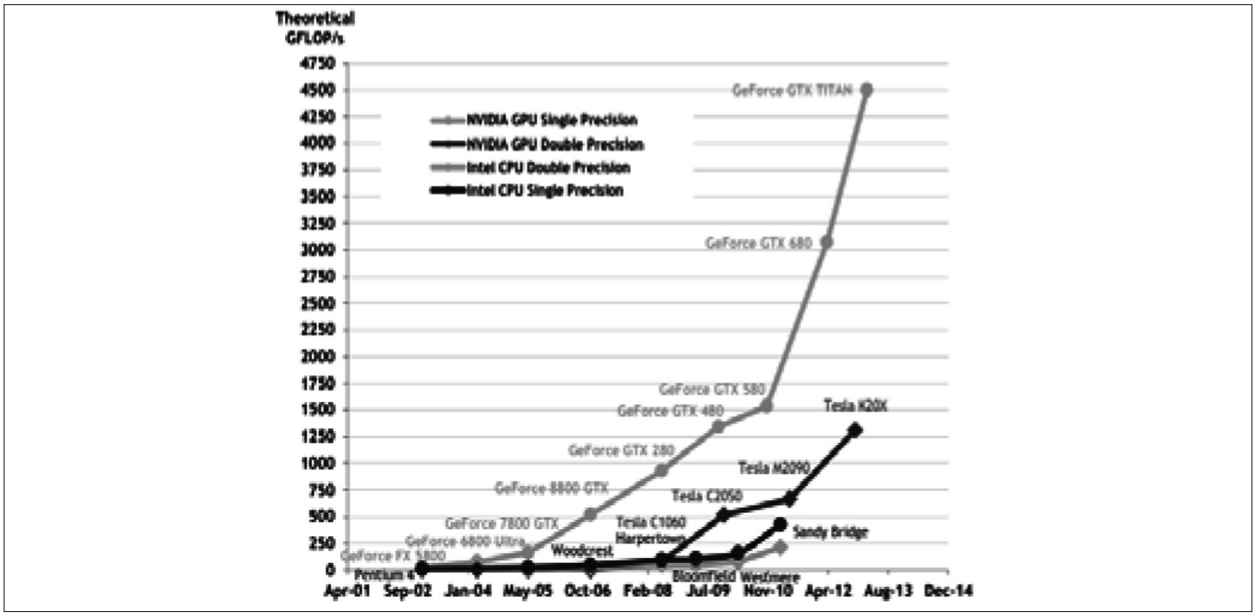
کد برنامه می‌تواند متشکل از چند بخش ردیفی و موازی باشد که به‌طور متناوب در کد برنامه تکرار می‌شوند. در شکل(۲)، اجرای همزمان یک برنامه در پردازنده اصلی و گرافیکی به نمایش گذاشته شده است[۱].

در این شکل برنامه و کد اصلی به دو بخش موازی (سبز رنگ) و ردیفی(آبی رنگ) تقسیم شده است و

21- Host Code
22- Device Code



شکل ۲: اجرای موازی کد در پردازنده گرافیکی



شکل ۳: مقایسه توان پردازش محاسبات ممیز شناور در پردازنده‌های اصلی و گرافیکی[۴]

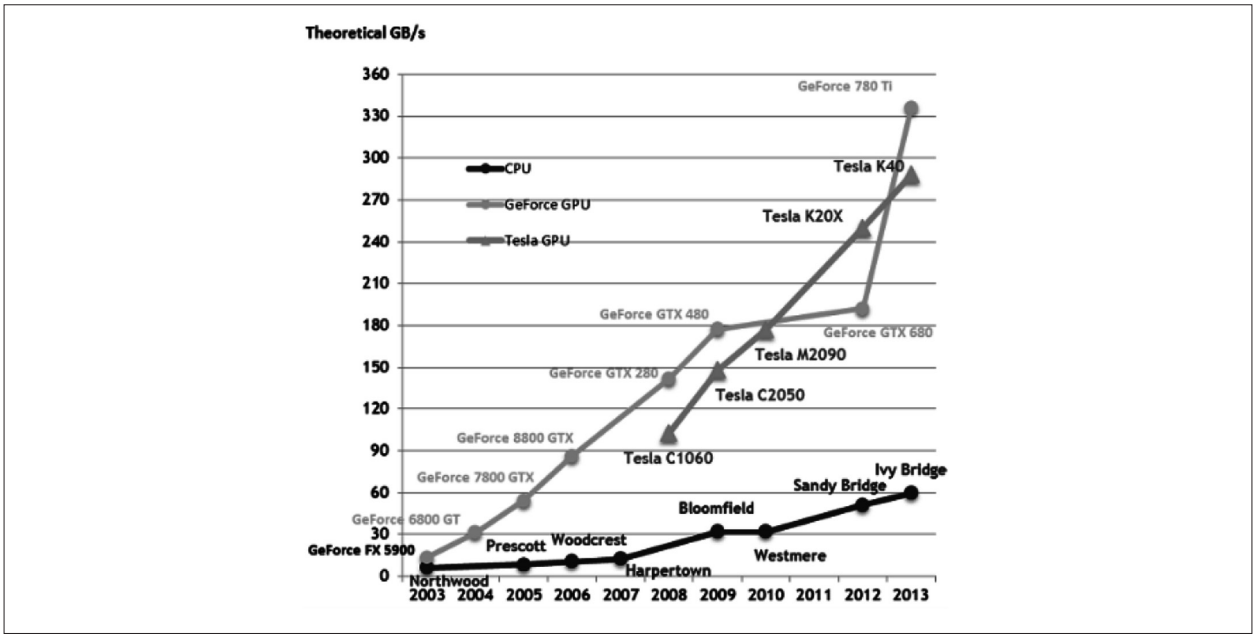
همان‌گونه که در شکل نشان داده شده است در ابتدا بخش ردیفی در پردازنده اصلی اجرا شده و سپس بخش موازی از طریق پردازنـــده اصلی و از طریق پردازنده گرافیکی به اجرا گذاشته می‌شـــود و در نهایت کد ردیفی در پردازنده اصلی اجرا می‌شود.

۲-۲-۲ مقایســه توان پردازشی پردازنده گرافیکی و اصلی

برای ســـنجش کارایی پردازنده‌های اصلی یا گرافیکی معیارهای مختلفی وجود دارد که در اینجا دو معیار اصلی توانایی محاســـبات[۲۳] و پهنای باند انتقال[۲۴] داده‌ها استفاده شده است. توان محاســـباتی به‌طور معمول با تعداد ممیز شـــناوری[۲۵] که یک پردازنده می‌تواند در واحد زمان اجرا نماید محاســـبه می‌شـــود و معمولاً بر حســـب تعداد گیگا عملیات در واحد زمان[۲۶] ســـنجیده می‌شـــود که هر چقدر این مقدار بیشـــتر باشـــد نشـــانه خوبی برای کارایی یک پردازنده به شمار می‌رود. در شکل(۳)، تعداد عملیات ممیز شناور دو نوع پردازنده گرافیکی شرکت انویدیا با دو نوع پردازنده غیرگرافیکی شـــرکت اینتل در ۱۱ ماه اول ســـال ۲۰۱۳ به نمایش گذاشته شده است. تجزیه و تحلیل نمودار نشـــان می‌دهد که روند افزایش محاسبات ممیز شناور در پردازنده‌هـــای گرافیکی به مراتب بیشـــتر از پردازنده‌های اصلی و غیر گرافیکی است.

نتایج بررســـی‌ها نشان می‌دهد که در یک سال توانایی تعداد ممیز شـــناور در پردازنده‌هـــای اصلی تغییر زیادی نداشـــته اســـت اما در این مدت معماری پردازنده گرافیکی با رشد بسیار زیاد توانسته توانایی انجام محاسبات ممیز شناور خود را به شدت افزایش دهد[٤]:

23- Peak computational performance
24- Memory bandwidth
25- Floating point
26- Gfloat per second



شکل ۴: مقایسه پهنای باند انتقال داده در محاسبات مورد استفاده پردازنده‌های اصلی و گرافیکی[۴]

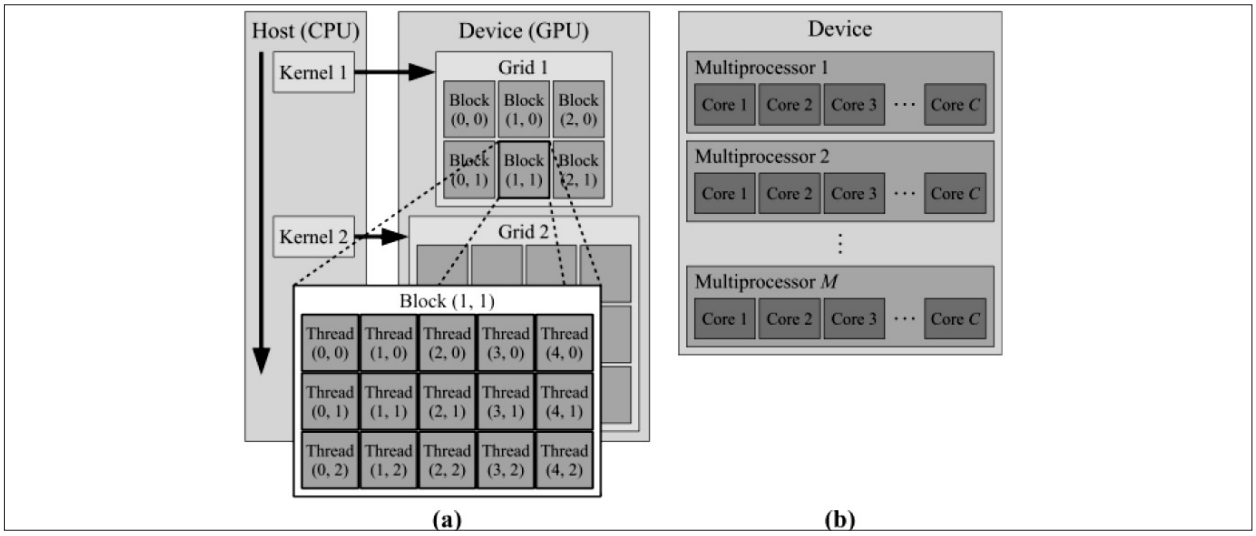
انجام محاسبات ممیز شناور به تنهایی نمی‌تواند کارایی یک پردازنده را مشخص نماید زیرا توانایی محاسبات یک پردازنده به عامل مهم پهنای باند گذرگاه حافظه‌ها نیز بستگی بالای دارد. یک پردازنده گرافیکی هر چقدر دارای توانایی بالاتری در محاسبات اعشاری باشد ولی خطوط ارتباطی آن با پردازنده اصلی ضعیف باشد نمی‌تواند از کارایی بالایی برخوردار باشد زیرا پهنای باند بالا بین حافظه پردازنده اصلی و گرافیکی به عنوان یک پل انتقال داده عمل می‌نماید که هر چقدر عرض آن بیشتر باشد در یک زمان می‌تواند داده‌های بیشتری را بین دو پردازنده جابجا نمود. در شکل(٤)، یک مقایسه بین میزان پهنای باند حافظه به‌کار رفته در پردازنده‌های اصلی و گرافیکی و هسته‌های آن‌ها در بین سال‌های ۲۰۰۳ تا ۲۰۱۳ نمایش داده شده است. در این نمودار پهنای دو پردازنده گرافیکی و یک پردازنده اصلی در طی زمان ۱۰ ساله نمایش داده شده است. تجزیه و تحلیل نمودار مورد نظر نشان می‌دهد که میزان پهنای باند محاسبات در پردازنده‌های گرافیکی بیشترین رشد ممکن را از خود نشان می‌دهند[٤]:

۲-۲-۳ چارچوب کودا

در سال ۲۰۰۷ انویدیا شرکت تولید کننده پردازنده‌ها و کارت‌های گرافیکی اولین نسخه از یک چارچوب نرم‌افزاری تحت نام چارچوب کودا[۲۷] را برای برنامه‌نویسی موازی به کمک پردازنده‌های گرافیکی این شرکت ارایه نمود. چارچوب کودا یک لایه نرم افزاری و شامل مجموعه‌ای از دستورات کتابخانه‌ای است. با استفاده از این چارچوب برنامه‌نویسان قادر خواهند بود برنامه‌های با زبان‌های رایج به‌ویژه VC++ را برای موازی‌سازی و اجرای الگوریتم‌های پیچیده توسعه دهند. کودا یک زبان برنامه‌نویسی محسوب نمی‌شود بلکه یک واسط بین برنامه‌نویس و پردازنده گرافیکی محسوب می‌شود. چارچوب کودا برای موازی‌سازی داده‌ها و الگوریتم‌های مختلف از واحدهای نرم‌افزاری ریسه، بلوک و توری استفاده می‌کند. هر ریسه می‌تواند بر روی یک هسته به صورت موازی اجرا شود یا مجموعه‌ای از ریسه‌ها بر روی یک هسته زمانبندی شوند. ریسه‌های واحدهای اجرای هسته می‌باشند به گونه‌ای که هر ریسه یک نسخه از هسته را اجرا می‌نماید. ریسه‌ها در واحدهای بزرگ‌تری مانند بلوک زمانبندی می‌شوند و مجموعه‌ای از ریسه‌ها در یک بلوک تعریف می‌شوند و به کمک حافظه اشتراکی قادر به ارتباط با هم می‌باشند.

ریسه‌های موجود در یک بلوک در صورتی که انتقال

27- CUDA framework



شکل ۵: اجزای اصلی به‌کار رفته در چارچوب کودا [۸ و ۷]

متـــن اتفاق بیفتد بلوک داده‌های خـــود را در حافظه محلی ذخیره می‌نمایند تـــا دوباره بتوانند عملیـــات موازی قبلی خود را انجام دهند. توری یک واحد اجرا برای راه‌اندازی[۲۸] هســـته‌ها به شمار می‌رود و هر هســـته توسط یک توری به اجرا گذاشته می‌شـــود که شامل مجموعه‌ای از بلوک‌ها می‌باشـــد[٦]. در کـــودا یک مفهـــوم کلیدی به نـــام جریان چند پردازندگی[۲۹] نیز وجـــود دارد که به معنی مجموعه‌ای از هســـته‌های کودا اســـت که در خدمت یک فرایند موازی قرار گرفته‌اند که در ادامه بیشـــتر توضیح داده می‌شـــود. در شـــکل(۵)، در تصویر a ســـاختار توری، بلوک و ریسه به صـــورت دو بعـــدی و در تصویر b چنـــد جریان چند پردازندگی در کودا نمایش داده شده است[۸ و ۷].

ساختار ریسه‌ها در حالتی که دو بعدی درون بلوک‌ها قرار گرفته باشند دارای دو شماره شناسه افقی و عمودی در راســـتای هر بلوک می‌باشند و این شماره ریسه‌ها فقط درون یک بلوک معتبر اســـت و محلی محسوب می‌شوند. می‌تـــوان بـــه کمک شـــماره‌های افقـــی و عمـــودی درون بلوکی هر ریســـه و ابعاد افقی و عمودی هر بلوک شماره منحصربه‌فردی را برای ریسه‌های متعلق به یک بلوک ارایه نمود به گونه‌ای که هیچ دو ریســـه در کل یک توری دارای یک شـــماره یکسان نباشـــند. در مورد بلوک‌های موجود در یک توری در حالت دوبعدی وضعیت مانند ریســـه‌های دوبعدی اســـت و هر بلوک دارای دو شماره انحصاری و محلی درون توری اســـت که به کمک ابعاد توری و شماره بلوکی می‌توان شماره منحصربه‌فرد بلوک‌های یک توری را محاسبه نمود[۹].

۲-۳ کارهـــای مرتبط با مـــوازی ســـازی الگوریتم‌های تکاملی

افزایـــش دقـــت و کاهش زمـــان اجـــرای الگوریتم‌های تکاملی، هدف اصلی در هنگام حل مسائل بهینه‌سازی است که ارائه راهکار موازی‌سازی در چارچوب کودا دسترسی به این هدف را امکان‌پذیر کرده است. الگوریتم هوش دسته جمعی ذرات یک تکنیک بهینه‌سازی مبتنی بر قوانین احتمال و رفتار جاندارانی اســـت که در دســـته‌های بزرگ زندگی می‌کنند. الگوریتم ذرات یک نمونه از رفتار دســـته جمعی و تبادل اطلاعات بین اجزا جمعیت محسوب می‌شود. در این الگوریتم هر جواب مســـئله در قالب ذرات مدل‌سازی شده و هر ذره دارای یک ســـرعت در فضای جستجوی مسئله اســـت که در هر تکرار این ســـرعت به کمک بردار سرعت موقعیت ذرات به روزرســـانی می‌شود. در سال‌های اخیر پژوهش‌هایی برای اجرای الگوریتم‌های تکاملی به‌صورت موازی ارائه شده است.

28- Launch
29- Stream Multiprocessor(SM)

فین گلر و همکارانش در سال ۲۰۱٤ برای حل مسئله سخت کوله‌پشتی از نسخه موازی کلونی مورچه‌ها در چارچوب کودا استفاده نمودند. در روش پیشنهادی آن‌ها، از مجموعه‌ای از هسته‌ها نظیر هسته اضافه کردن تصادفی اشیاء به هر کوله، یافتن بهترین ترکیب کوله ها با الگوریتم کلونی مورچه‌ها جهت انتخاب بهترین حالت این مسئله استفاده شده است. نتایج پژوهش آن‌ها به‌طور موثر مسئله کوله‌پشتی را حل نموده و زمان حل آن به مراتب از نسخه ردیفی کمتر است[۱۰].

جوهنگ و همکارانش در سال ۲۰۱٤ یک نسخه موازی از الگوریتم زنبور عسل با استفاده از معماری کودا ارائه نمودند. در این الگوریتم ریسه‌ها در یک بلوک به چندین کلونی دسته‌بندی می‌شوند و هر ریسه به یک زنبورعسل برای جستجوی راه‌حل اختصاص داده می‌شود. الگوریتم پیشنهادی یک بلوک را به کلونی‌های مختلف با شناسه ریسه تقسیم می‌کند و به‌طور مستقل الگوریتم زنبورها را اجرا می‌کند. روش پیشنهادی آن‌ها از الگوریتم استاندارد غیرموازی زنبور حداقل ۱۳بار و در ۹تابع مختلف بهینه‌سازی سریع‌تر عمل می‌نماید[۱۱].

فی ری رو و همکارانش در سال ۲۰۱۵ یک نسخه موازی و بهبود یافته از الگوریتم گرم و سرد به کمک هسته‌های موازی پردازنده گرافیکی و در چارچوب کودا ارائه نمودند. آن‌ها به کمک یک هسته هر وضعیت الگوریتم سرد و گرم را شبیه‌سازی نمودند به شکلی که در هر تکرار یک وضعیت از فضای مسئله ایجاد می‌شود و مجموعه‌ای از وضعیت‌های همسایه توسط مجموعه‌ای از ریسه‌ها زمان‌بندی شده تا بهترین وضعیت یا جواب مسئله استخراج شود[۱۲].

سرآپیااو و همکارانش در سال ۲۰۱٦ برای خوشه‌بندی دقیق‌تر داده‌ها از یک الگوریتم ترکیبی که متشکل از الگوریتم خوشه‌بندی Kmeans و K-Harmonic به همراه یک نسخه تکاملی از الگوریتم ماهی می‌باشد در محیط موازی پردازنده گرافیکی استفاده نمودند. روش آن‌ها علاوه بر دقت مناسب، به علت استفاده از معماری موازی کودا دارای دقت و سرعت مناسب‌تری نسبت به الگوریتم خوشه‌بندی Kmeans می‌باشد[۱۳].

مایسون و همکارانش در سال ۲۰۱۷ برای کاهش زمان مذاکرات سطح در رایانش ابری و افزایش توان تولید و بهینه‌سازی، الگوریتمی براساس موازی سازی ذرات ارائه کردند که براساس آن سرعت ۳۰درصد و بهره‌وری ۱۵درصد بهبود یافت[۱٤].

لای و همکارانش در سال ۲۰۱۸ روشی به منظور بهبود الگوریتم ذرات با استفاده از موازی سازی جمعیت چندگانه ارائه دادند. بسیاری از توابع در مقیاس بزرگ با استفاده از یک الگوریتم ذرات به‌صورت موازی قابل حل است و برای به دست آوردن عملکرد بهتر، منطقه جستجو به‌طور مساوی و به‌صورت پویا تقسیم گردید[۱۵].

۳- روش پیشنهادی

موازی سازی رویکردی است که در آن هر ذره یا عضو جمعیت اولیه دارای امکانات سخت‌افزاری مجزا باشد و به‌عبارت بهتر هر عضو جمعیت اولیه توانایی اجرا در یک هسته پردازشی را داشته باشد.

به علت محدودیت هسته‌های پردازشی پردازنده اصلی، این پردازنده توانایی در اختیار گذاشتن یک هسته برای هر عضو جمعیت را ندارد. گام‌های اصلی موازی‌سازی الگوریتم‌های تکاملی شامل: ۱- ایجاد جمعیت اولیه، ۲- پردازش اعضای جمعیت توسط هسته، ۳- یافتن بهترین اعضای جمعیت می‌باشد.

الگوریتم‌های تکاملی از جمله بهینه‌سازی ذرات برای یافتن بهینه مسائل نیاز به یک جمعیت به اندازه مشخص دارند تا در فضای جستجوی مسئله به دنبال بهینه سراسری تابع هدف بگردند و هر چقدر این توابع شکل پیچیده‌تر داشته باشند نیاز است که اندازه جمعیت به قدر کافی بزرگ در نظر گرفته شود تا کل فضای جستجوی تابع هدف برای یافتن بهینه سراسری مورد پیمایش قرار گیرد. در الگوریتم

بهینه‌سازی ذرات در صورتی‌که اندازه جمعیت اولیه (اندازه داده مسئله) کوچک باشد فضای جستجوی مسئله به‌خوبی برای یافتن بهینه مسئله مورد جستجو قرار نمی‌گیرد. لذا نیاز است که اندازه جمعیت را افزایش داده تا کل فضای مسئله پوشش داده شود و در حین حال برای جلوگیری از کند شدن الگوریتم پیشنهادی آن را به شکل موازی در پردازنده گرافیکی و توسط چارچوب کودا به اجرا می‌گذاریم. هر الگوریتمی را نمی‌توان به شکل موازی به اجرا گذاشت و باید این الگوریتم دارای ساختار و عملکردی باشد که به راحتی موازی‌سازی شود. الگوریتم بهینه‌سازی ذرات یک الگوریتم مبتنی بر جمعیت است که هر عضو جمعیت آن می‌توان فضای جستجوی تابع هدف را برای یافتن بهینه به کمک بهترین عضو سراسری و بهترین سابقه حرکت خود مورد جستجو قرار دهد و این سازوکار اجازه می‌دهد که الگوریتم ذرات به صورت موازی قابل اجرا باشد.

روش پیشنهادی برای موازی‌سازی الگوریتم ذرات بهبود یافته دارای مراحل اصلی زیر است:

• ذرات در قالب دو بردار سرعت و موقعیت کد می‌شوند و هر قسمتی از بردار موقعیت و سرعت به یک ذره اختصاص دارد.

• پارامترهای اولیه کودا مانند تعداد بلوک و ریسه‌های هر بلوک و پارامترهای الگوریتم پیشنهادی مانند تعداد ذرات و تعداد تکرار مقداردهی اولیه می‌شوند.

• مقادیر بردار موقعیت و سرعت ذرات به صورت تصادفی مقداردهی اولیه می‌شود.

• حافظه مناسب برای بردار سرعت و موقعیت ذرات در کودا تخصیص داده می‌شود.

• کپی نمودن دو بردار سرعت و موقعیت از فضای حافظه میزبان به حافظه سراسری دستگاه

• اجرای هسته‌ای که دارای کد موازی‌سازی الگوریتم ذرات پیشنهادی است. این هسته توسط مجموعه‌ای از ریسه‌های موجود در یک بلوک به اجرا گذاشته می‌شود.

• تکرار متوالی هسته برای تغییر موقعیت ذرات در فضای جستجوی مسئله

• به روزرسانی بهترین ذره سراسری در هر اجرای هسته

• انتقال بهترین ذره سراسری از حافظه سراسری دستگاه به حافظه اصلی میزبان

• ارزیابی زمان اجرا

• آزاد نمودن حافظه‌های به‌کار رفته در تعریف بردار سرعت و موقعیت ذرات

در روش پیشنهادی برای موازی‌سازی الگوریتم بهینه‌سازی ذرات می‌توان دو آرایه $(v_D^n$ و ... و v_2^n و v_1^n و ... و v_D^1 و ... و v_2^1 و $v_1^1)$ و $(f^n$ و x_D^n و ... و x_2^n و x_1^n و f^1 و x_D^1 و ... و x_2^1 و $x_1^1)$ را که به ترتیب سرعت و موقعیت کل ذرات را در خود نگهداری می‌نمایند، تعریف نمود. سپس توسط دستور cudaMalloc حافظه مورد نیاز آن‌ها تخصیص داده می‌شود. در این ساختار بردارهای مورد نظر n تعداد ذرات، D ابعاد تابع ارزیابی، f^i شایستگی ذره iام، v_j^i مولفه jام سرعت ذره iام و x_j^i مولفه jام موقعیت ذره iام می‌باشد. اندازه بردار موقعیت و سرعت به ترتیب برابر $n \times (D+1)$ و $n \times D$ است.

سپس مقادیر این دو بردار در حافظه اصلی به صورت تصادفی مقداردهی اولیه می‌شود و با استفاده از دستور cudaMemcpy و با استفاده از سوئیچ cudaMemcpyHostToDevice مقادیر دو آرایه از حافظه اصلی میزبان به حافظه سراسری پردازنده گرافیکی منتقل می‌شود.

در ادامه هر ریسه یک نسخه از هسته که شامل دستورات موازی الگوریتم پیشنهادی برای تعیین موقعیت جدید ذرات است را به اجرا می‌گذارند. برای اجرای هسته تغییر موقعیت ذرات به تعداد اعضای جمعیت اولیه از طریق کودا ریسه در نظر گرفته می‌شود. بعد از تعیین موقعیت هر ذره در پایان تکرار فعلی نیاز است که از طریق فراخوانی هستهٔ دوم بهترین ذره سراسری جمعیت محاسبه شود و موقعیت آن به روزرسانی شود. در این حالت هستهٔ قبلی از

جدول ۱: توابع ارزیابی به‌کار رفته در شبیه‌سازی

تابع ارزیابی	ضابطه
Sphere	$f = \sum_{i=1}^{n} x_i^2$
Rosenbrock	$f = \sum_{i=1}^{n-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$
Griewank	$f = 1 + \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)$
Rastrigin	$f = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i))$

به روز رسانی بهترین ذره سراسری از رابطه(۱) شماره شناسه و منحصربه‌فرد خود را به‌دست می‌آورد و از طریق این شماره شناسایی می‌تواند به خانه‌های i تا i+D+1 بردار موقعیت و i+D بردار سرعت که درون حافظه سراسری دستگاه قرار دارند دسترسی پیدا نمایند:

i = blockIdx.x * blockDim.x + threadIdx.x (۱)

بعد از اتمام تکرار الگوریتم بهترین ذره سراسری که در حافظه سراسری دستگاه قرار دارد به کمک دستور cudaMemcpy و با سوئیچ cudaMemcpyDeviceToHost به حافظه اصلی میزبان انتقال پیدا نموده و به خروجی برده می‌شود و در نهایت حافظه‌های به‌کار رفته توسط کودا آزاد می‌شود. در شکل(٦)، شبه کد روش پیشنهادی برای موازی‌سازی و تسریع الگوریتم بهینه‌سازی ذرات نشان داده شده است.

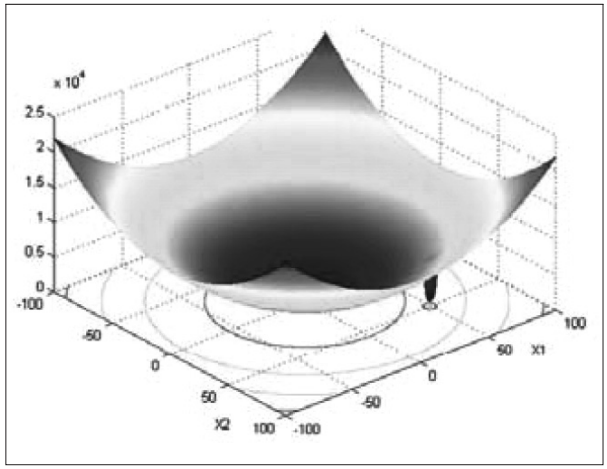
۴- تجزیه و تحلیل

برای سنجش دقت و سرعت همگرایی الگوریتم‌های تکاملی از جمله الگوریتم بهینه‌سازی ذرات توابعی ریاضی تحت نام توابع هم‌سنجی وجود دارند که به عنوان تابع هدف مسئله در نظر گرفته می‌شود. در این توابع هدف که معرف یک مسئله بهینه‌سازی می‌باشد یافتن کمینه سراسری هدف نهایی الگوریتم تکاملی مورد نظر است[١٦].

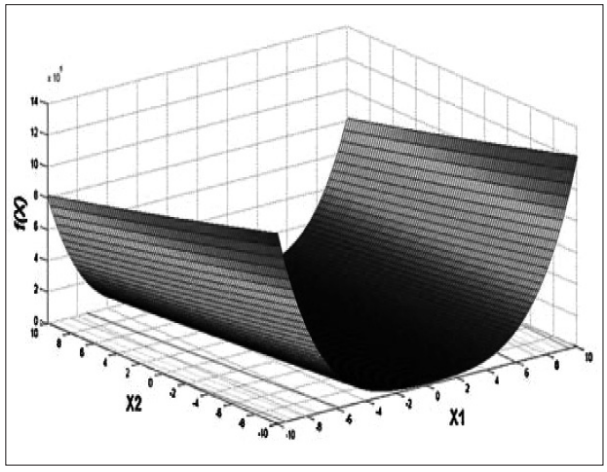
در جدول(۱) تعدادی از توابع ارزیابی که برای سنجش دقت و سرعت الگوریتم پیشنهادی به‌کار گرفته می‌شود همراه با ضابطه و مقادیری ازمتغیرها که تابع به ازای آن

```
1. Initialization
   Set Itermax, nPop, C1,  C2, Inertia factor
2. Create Population
   Create positon and velocity vectors in host randomly
3. Memory Allocation Device
   Define dev_positon and dev_velocity in CUDA
4. Copy Population from Host to Device
   Copy positon and velocity vectors to dev_positon and dev_velocity
   by cudamemcpy(HostToDevice)
5. launch Kernel
   id = blockIdx.x * blockDim.x + threadIdx.x
   Update dev_velocity by threads then Update dev_ positon by threads
   V_id(t+1) = ωV_id(t) + c1r1(P^id_p − X_id(t)) + c2r2(P^id_g − X_id(t))
   X_id(t+1) = V_id(t+1) + X_id(t)
  If bestSoulation< globalBest then
    globalBest= bestSoulation
  end
6. Copy Population from Device to Host
   Show globalBest and  free CUDA memory
```

شکل ۶: شبه کد روش پیشنهادی در تسریع الگوریتم ذرات



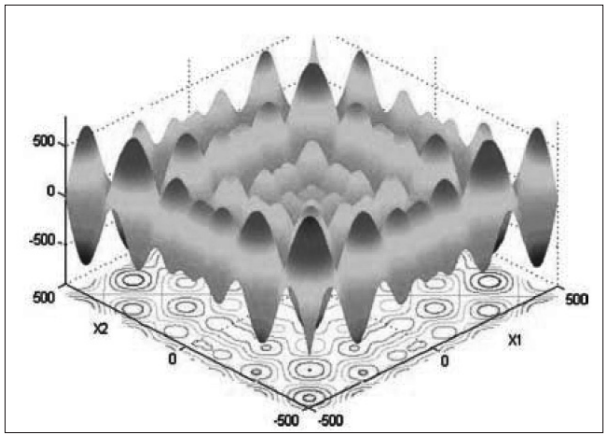
شکل ۷: تابع ارزیابی Sphere



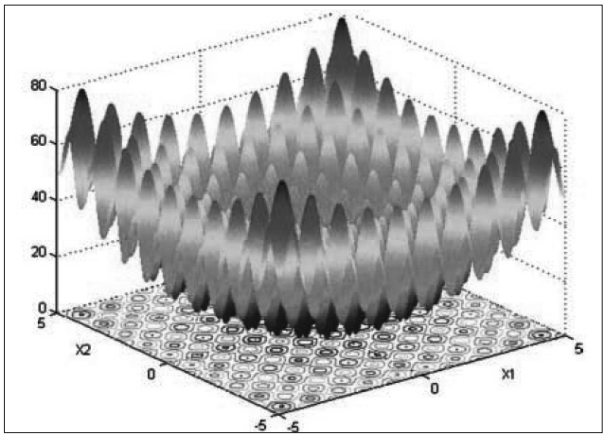
شکل ۸: تابع ارزیابی Rosenbrock

حافظه ریسه‌ها خارج شده و اقدام به اجرای موازی هستهٔ دوم می‌نمایند.

هر ریسه برای اجرای هسته به روز رسانی موقعیت یا



شکل ۹: تابع ارزیابی Griewank



شکل ۱۰: تابع ارزیابی Rastrigin

کمینه می‌شود نشان داده شده است[۱۷].

در شـــکل(۷)، (۸)، (۹) و (۱۰) نمودار ســـه‌بعدی توابع ارزیابـــی Griewank ،Rosenbrock ،Sphere و Rastrigin نمایش داده شده است:

برای بهبود ســـرعت الگوریتم پیشنهادی نیاز است که در مرحله دوم پژوهش آن را به شکل موازی در چارچوب کودا پیاده‌ســـازی نموده و با محاسبه زمان اجرای نسخه ردیفی و موازی آن کارایی الگوریتم را مشخص می‌نمائیم. در این قسمت نیز برای سنجش سرعت نسخه موازی نسبت به نســـخه ردیفی از توابع ارزیابی Sphere، Rosenbrock، Griewank، Sum Squares، Beale، Easom، Three Hump و Rastrigin اســـتفاده می‌شـــود. دو معیار مهم زمان اجرا و شتاب به‌طور معمول برای سنجش کارایی الگوریتم‌های موازی در کودا به‌کار گرفته می‌شـــود.در رابطه(۲)، شتاب پردازنده گرافیکی به صورت زمان اجرا در پردازنده اصلی به گرافیکی تعریف شده است:

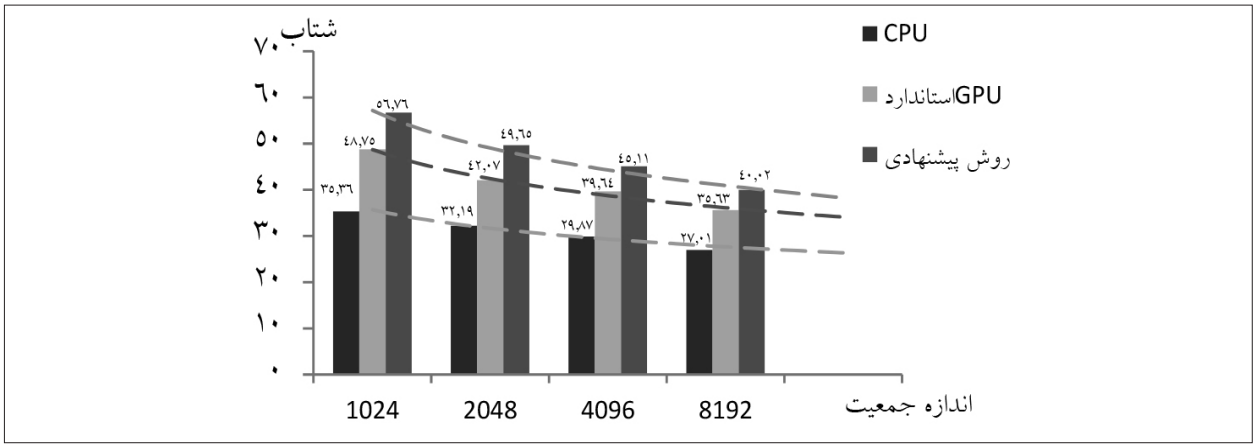
$$Speedup = \frac{Runtime\ CPU}{Runtime\ GPU} \qquad (۲)$$

برای شبیه‌سازی و پیاده‌سازی‌ها از سخت افزاری دارای پردازنـــده مرکـــزی Intel i5-3337U CPU @ 1.80GHz حافظـــه RAM برابـــر ٤ گیـــگا بایت و پردازنـــده گرافیکی Geforce 710M بـــا حافظه یک گیگا بایت اســـتفاده شـــده اســـت. برای محاسبه میزان شـــتاب الگوریتم پیشنهادی که همان نســـبت زمان اجرا در پردازنده اصلی به گرافیکی است جمعیت اولیه را به عنوان مسئله ورودی متغیر گرفته و بر تعداد آن افزوده ســـپس شـــتاب را پی‌درپی محاسبه می‌نمائیم.

در این قسمت برای هر کدام از توابع ارزیابی ۵۰ آزمایش مختلـــف را در کـــودا انجام داده و زمان اجـــرای الگوریتم مـــوازی ذرات و اســـتاندارد(موازی معمولی) و به‌صورت ردیفی را محاســـبه نموده سپس متوسط زمان را محاسبه و از آن برای محاسبه شتاب متوسط در هر کدام از توابع ارزیابی استفاده نموده‌ایم. اندازه جمعیت در الگوریتم‌های تکاملی نقش مهمی در دقت و البته سرعت اجرای آن‌ها دارد و هر چقدر که اندازه جمعیت اولیه آن‌ها بزرگ باشـــد دقت الگوریتـــم افزایش و البته زمان اجـــرا نیز افزایش که باعث کاهش سرعت الگوریتم مورد نظر می‌شود.

نمودار شتاب متوسط در جمعیتی با اندازه‌های ۱۰۲٤، ۲۰٤۸، ٤۰۹٦ و ۸۱۹۲ نشـــان می‌دهـــد که کارایی پردازنده گرافیکی با افزایش اندازه داده‌ها بیشتر می‌شود و افزایش اندازه داده‌ها یا همان جمعیت اولیه می‌تواند شتاب مناسبی به الگوریتم پیشنهادی دهد.

تجزیه و تحلیل نمودار شـــتاب بر حسب اندازه جمعیت اولیه نشـــان می‌دهد که زمان در هر دو پردازنده اصلی و گرافیکی برای اجرای الگوریتـــم ذرات افزایش یابد اما این میزان افزایش در سمت پردازنده اصلی به مراتب بیشتر از پردازنده گرافیکی است و همین علت باعث می‌شود نسبت زمـــان اجرا در پردازنده اصلی بـــه گرافیکی مرتباً افزایش



شکل ۱۱: افزایش شتاب با افزایش اندازه جمعیت

جدول ۲: نتایج زمان اجرا

نام تابع	تعداد ذرات	زمانCPU (ثانیه)	زمانGPU (ثانیه)	زمان روش پیشنهادی (ثانیه)
Sphere	۱۰۰۰	۳۸,۹۲۹	۹,۴۵۶	۷,۲۹۵
	۲۰۰۰	۶۴,۳۲۵	۱۵,۴۶۱	۱۲,۷۳۷
	۳۰۰۰	۱۰۹,۲۳۹	۶۷,۴۰۴	۲۴,۱۷۳
Rosenbrock	۱۰۰۰	۸۲,۶۵۲	۳۳,۵۰۱	۳۱,۰۲۲
	۲۰۰۰	۱۴۸,۲۸	۵۵,۰۳۱	۵۲,۳۳۷
	۳۰۰۰	۲۳۵,۰۴۲	۱۲۴,۱۷۱	۸۱,۳۸۷
Griewank	۱۰۰۰	۴۹,۹۰۱	۱۶,۱۳۵	۱۴,۰۱۹
	۲۰۰۰	۸۴,۲۳۰	۲۶,۵۶۴	۲۳,۷۵۶
	۳۰۰۰	۱۲۴,۵۲۸	۶۲,۱۷۱	۳۸,۶۷۸
Rastrigin	۱۰۰۰	۶۱,۶۵	۳۴,۶۰۱	۲۳,۸۴۱
	۲۰۰۰	۸۷,۸۱۳	۵۸,۲۱۶	۳۹,۸۲۱
	۳۰۰۰	۱۹۱,۸۱	۷۳,۹۲۷	۵۸,۰۱۸

می‌یابد به نحوی که باعث می‌شود میزان شتاب با افزایش جمعیت نیز روندی صعودی داشته باشد. به‌طور کلی، تجزیه و تحلیل خروجی‌های شبیه‌سازی و نتایج شتاب بر روی توابع ارزیابی مختلف نشان می‌دهد با افزایش اندازه جمعیت ذرات به عنوان داده ورودی مسئله زمان اجرا در پردازنده گرافیکی و اصلی مرتب افزایش می‌یابد و این افزایش به گونه‌ای است که نرخ افزایش زمان اجرا در پردازنده اصلی به مراتب بیشتر از پردازنده گرافیکی است و این باعث می‌شود نسبت زمان اجرا در پردازنده اصلی به پردازنده گرافیکی با افزایش اندازه جمعیت افزایش یابد.

به عبارت بهتر کارایی پردازنده گرافیکی زمانی بیشتر خود را نشان می‌دهد که داده‌های ما به‌قدر کافی بزرگ باشند.

مقایسه اجرای الگوریتم بهینه‌سازی ذرات بر روی CPU، بر روی GPU و روش پیشنهادی نشان‌دهنده بهبود روش پیشنهادی است. با افزایش تعداد ذرات در روش پیشنهادی، زمان کاهش می‌یابد.

نتیجه‌گیری

الگوریتم بهینه‌سازی ذرات یک الگوریتم تکاملی با رویکرد هوش دسته جمعی است که مجموعه ذرات می‌تواند به‌صورت موازی فضای جستجوی مسئله را مورد جستجو قرار دهند با این وجود در الگوریتم بهینه‌سازی ذرات با افزایش اندازه جمعیت اولیه زمان اجرای آن بیش از اندازه زیاد می‌شود لذا در این مقاله یک روش موازی‌سازی مبتنی بر کودا برای بهبود سرعت الگوریتم بهینه‌سازی ذرات ارایه شد. نتایج پیاده‌سازی روش پیشنهادی در پردازنده گرافیکی با استفاده از چارچوب کودا نشان می‌دهد که افزایش اندازه جمعیت، زمان اجرای الگوریتم پیشنهادی و بهینه‌سازی ذرات را افزایش می‌دهد اما این افزایش در پردازنده گرافیکی نسبت به پردازنده اصلی اندک است و این باعث می‌شود شتاب[30] که نسبت زمان اجرا در پردازنده گرافیکی به پردازنده است با افزایش اندازه جمعیت افزایش یابد.

30- Speedup

در پژوهـــش آتـــی قصد داریـــم از چارچـــوب موازی کودا برای تســـریع سایر الگوریتم‌های هوش دسته جمعی اســـتفاده نمائیم تا جواب‌های بهینه یک مســـئله با سرعت بیشتری محاسبه شوند.

مراجع

1-P.Mark. Wachowiak, C.Mitchell. Timson, and J .David. DuVal. “Adaptive Particle Swarm Optimization with Heterogeneous Multicore Parallelism and GPU Acceleration” ,Journal of IEEE Transactions on Parallel and Distributed Systems Published by the IEEE Computer Society, 2017, pp:1-11.

2-M.Couceiro, and P.Ghamisi, “Particle Swarm Optimization”. Chapter Fractional Order Darwinian Particle Swarm Optimization,Part of the series SpringerBriefs in Applied Sciences and Technology, 2015, pp:1-11.

3-J.Kołodziejczyk, D.Sychel, AND A.Bera, “Improved CUDA PSO Based on Global Topology”, International Conference on Artificial Intelligence and Soft Computing, Artificial Intelligence and Soft Computing, 2017, pp: 347-358.

4-S.Mittal., and S.J.Vetter., “A Survey of CPU-GPU Heterogeneous Computing Techniques”, Journal ACM Computing Surveys (CSUR), pp:1-35, 2015.

5-R,A Brodtkorb., T.R.Hagen., and Sætra.M.L, “Graphics processing unit (GPU) programming strategies and trends in GPU computing”, Journal of Parallel and Distributed Computing, 2013, pp:4-13.

6-E,Wynters.” Parallel particle swarm optimization can solve many optimization problems quickly on GPUS”, Journal of Computing Sciences in Colleges, 2018, pp:114-123.

7-J.V. de Oliveira, A.Szabo, “Particle Swarm Clustering in clustering ensembles”, Applied Soft Computing Journal, 2017, pp: 141-153

8-J., Cheng, M., Grossman, & T. McKercher, “Professional Cuda C Programming”. Publisher: Wiley John,2014, pp:1-526.

9-Nvidia, C. U. D. A. “CUDA C Programming Guide 5.5”, NVIDIA Corporation, 2013, pp: 4-16.

10-H. Fingler, E. N. Cáceres, H. Mongelli, and S. W. Song, “A CUDA based Solution to the Multidimensional Knapsack Problem Using the Ant Colony Optimization,” Procedia Comput. Sci, 2014, pp:84–94.

11-L.Guo-Heng, S.-K.Huang, Y.-S.Chang, and S.-M.Yuan, “A parallel Bees Algorithm implementation on GPU,” J. Syst. Archit., 2014, pp:271–279.

12-C. Ferreiro, A. M., Garcia, J. A., Lopez-Salas, J. G., & Vazquez, “An efficient implementation of parallel simulated annealing algorithm in GPUs.Journal of Global Optimization,” 2015, pp:863–890.

13-A. B. S. Serapiao, G. S. Correa, F. B. Goncalves, and V. O. Carvalho, “Combining K-Means and K-Harmonic with Fish School Search Algorithm for data clustering task on graphics processing units,” Appl. Soft Comput, 2016, pp:290-304.

14-F. A.Maysoon , S. A .Entisar, R. J.Nicholas, “Automated Negotiation using Parallel Particle Swarm Optimization for Cloud Computing Applications” , International Conference on Computer and Applications (ICCA), 2017, pp:26-35.

15-X.Lai, & Y.Zhou, “An adaptive parallel particle swarm optimization for numerical optimization problems”, Neural Computing and Applications Journal, 2018, pp:1-19.

16-A. Manconi, E. Manca, M., Moscatelli, M., Gnocchi, Orro.A., G. Armano, & L. Milanesi, “G-CNV: a GPU-based tool for preparing data to detect CNVs with read-depth methods”. Frontiers in bioengineering and biotechnology, 2015, pp:1-14.

17-E. Mejia-Roa, D.Tabas-Madrid, J. Setoain, C.Garcia, F.Tirado, & A.Pascual-Montano, “NMF-mGPU: non-negative matrix factorization on multi-GPU systems”, BMC Bioinformatics (BMC BIOINFORMATICS) 2015, pp:1-12.